

Informatik 1

Wintersemester 2006/2007

Helmut Seidl

**Institut für Informatik
TU München**

0 Allgemeines

Inhalt dieser Vorlesung:

- Einführung in Grundkonzepte der Informatik;
- Einführung in Denkweisen der Informatik;
- Programmieren in Java :-)

Voraussetzungen:

Informatik Leistungskurs:

nützlich, aber nicht nötig ;-)

Kenntnis einer Programmiersprache:

nützlich, aber nicht nötig :-)

Eigener Rechner:

nützlich, aber nicht nötig :-)

Voraussetzungen:

Informatik Leistungskurs:

nützlich, aber nicht nötig :-)

Kenntnis einer Programmiersprache:

nützlich, aber nicht nötig :-)

Eigener Rechner:

nützlich, aber nicht nötig :-)

Abstraktes Denken:

unbedingt erforderlich !!!

Neugierde, technisches Interesse:

unbedingt erforderlich !!!

Unsere Philosophie:

Schreiben ist Macht

Eine alte Kulturtechnik:

- um Wissen haltbar zu machen;
- neue Denktechniken zu entwickeln ...

Schreiben als politisches Instrument:

- ⇒ um zu bestimmen, was Realität ist;
- ⇒ um das Handeln von Menschen zu beeinflussen ...
(Vorschriften, Gesetze)

Schreiben als praktisches Werkzeug:

- ⇒ um festzulegen, wie ein Vorgang ablaufen soll:
 - Wie soll das Regal zusammen gebaut werden?
 - Wie multipliziert man zwei Zahlen?
- ⇒ um das Verhalten komplexer mit ihrer Umwelt interagierender Systeme zu realisieren
(etwa Motorsteuerungen, Roboter)

⇒ Programmierung

1 Vom Problem zum Programm

Ein **Problem** besteht darin, aus einer gegebenen Menge von Informationen eine weitere (bisher unbekannte) Information zu bestimmen.

Ein **Algorithmus** ist ein exaktes **Verfahren** zur Lösung eines Problems, d.h. zur Bestimmung der gewünschten Resultate.



Ein Algorithmus beschreibt eine Funktion: $f : E \rightarrow A$,
wobei E = zulässige Eingaben, A = mögliche Ausgaben.



Abu Abdallah Muhamed ibn Musa al Chwaritzmi, etwa 780–835

Achtung:

Nicht jede Abbildung lässt sich durch einen Algorithmus realisieren!
(↑ **Berechenbarkeitstheorie**)

Das **Verfahren** besteht i.a. darin, eine Abfolge von **Einzelschritten** der Verarbeitung festzulegen.

Beispiel: Alltagsalgorithmen

Resultat	Algorithmus	Einzelschritte
Pullover	Strickmuster	eine links, eine rechts eine fallen lassen
Kuchen	Rezept	nimm 3 Eier ...
Konzert	Partitur	Noten

Beispiel: Euklidischer Algorithmus

Problem: Seien $a, b \in \mathbb{N}, a, b \neq 0$. Bestimme $\text{ggT}(a, b)$.

Beispiel: Euklidischer Algorithmus

Problem: Seien $a, b \in \mathbb{N}, a, b \neq 0$. Bestimme $\text{ggT}(a, b)$.

Algorithmus:

1. Falls $a = b$, brich Berechnung ab, es gilt $\text{ggT}(a, b) = a$.
Ansonsten gehe zu Schritt 2.
2. Falls $a > b$, ersetze a durch $a - b$ und setze Berechnung in Schritt 1 fort.
Ansonsten gehe zu Schritt 3.
3. Es gilt $a < b$. Ersetze b durch $b - a$ und setze Berechnung in Schritt 1 fort.

Eigenschaften von Algorithmen:

Abstrahierung: Allgemein löst ein Algorithmus eine **Klasse** von Problem-Instanzen. Die Anwendung auf eine **konkrete** Aufgabe erfordert Abstraktion :-)

Determiniertheit: Algorithmen sind im allgemeinen determiniert, d.h. mit gleichen Eingabedaten und gleichem Startzustand wird stets ein gleiches Ergebnis geliefert. (↑ **nichtdeterministische Algorithmen**, ↑ **randomisierte Algorithmen**)

Finitheit: Die Beschreibung eines Algorithmus besitzt endliche Länge. Die bei der Abarbeitung eines Algorithmus entstehenden Datenstrukturen und Zwischenergebnisse sind endlich.

Terminierung: Algorithmen, die nach endlich vielen Schritten ein Resultat liefern, heißen **terminierend**. Meist sind nur terminierende Algorithmen von Interesse. **Ausnahmen:** **Betriebssysteme, reaktive Systeme, ...**

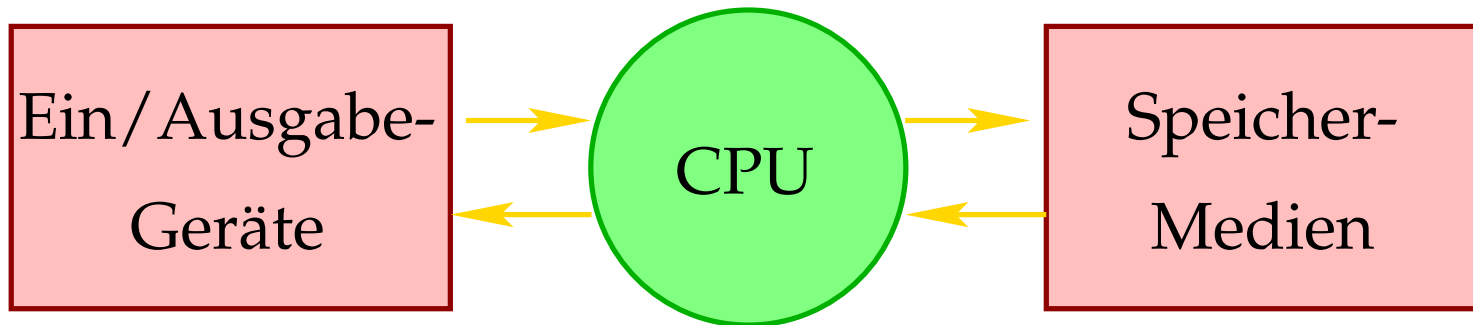
Ein **Programm** ist die **Formulierung** eines Algorithmus in einer **Programmiersprache**.

Die Formulierung gestattet (hoffentlich :-)) eine maschinelle Ausführung.

Beachte:

- Es gibt viele Programmiersprachen: **Java, C, Prolog, Fortran, Cobol**
- Eine Programmiersprache ist dann **gut**, wenn
 - **die Programmiererin** in ihr ihre algorithmischen Ideen **natürlich** beschreiben kann, insbesondere selbst später noch versteht, was das Programm tut (oder nicht tut);
 - **ein Computer** das Programm leicht verstehen und **effizient** ausführen kann.

Typischer Aufbau eines Computers:



Ein/Ausgabegeräte (= input/output devices) — ermöglichen Eingabe des Programms und der Daten, Ausgabe der Resultate.

CPU (= central processing unit) — führt Programme aus.

Speicher-Medien (= memory) — enthalten das Programm sowie die während der Ausführung benötigten Daten.

Hardware == physikalische Bestandteile eines Computers.

Merkmale von Computern:

Geschwindigkeit: schnelle Ausgeführt auch komplexer Programme.

Zuverlässigkeit: Hardwarefehler sind selten :-)
Fehlerhafte Programme bzw. falsche Eingaben sind häufig :-(

Speicherkapazität: riesige Datenmengen speicherbar und schnell zugreifbar.

Kosten: Niedrige laufende Kosten.

Algorithmen wie Programme **abstrahieren** von (nicht so wesentlichen) Merkmalen realer Hardware.

⇒ Annahme eines (nicht **ganz** realistischen, dafür exakt definierten) **Maschinenmodells** für die Programmierung.

Beliebte Maschinenmodelle:

Turingmaschine: eine Art Lochstreifen-Maschine
(Turing, 1936 :-)

Registermaschine: etwas realistischerer Rechner, allerdings mit
i.a. beliebig großen Zahlen und unendlich viel Speicher;

λ -Kalkül: eine minimale \uparrow funktionale Programmiersprache;

JVM: (Java-Virtual Machine) – die abstrakte Maschine für Java
(\uparrow Compilerbau);

...

Zur Definition eines Maschinenmodells benötigen wir:

- Angabe der zulässigen Datenobjekte/Speicherbereiche, auf denen Operationen ausgeführt werden sollen;
- Angabe der verfügbaren Einzelschritte / Aktionen / Elementaroperationen;
- Angabe der Kontrollstrukturen zur Angabe der beabsichtigten Ausführungsreihenfolgen.

Beispiel 1: Turing-Maschine

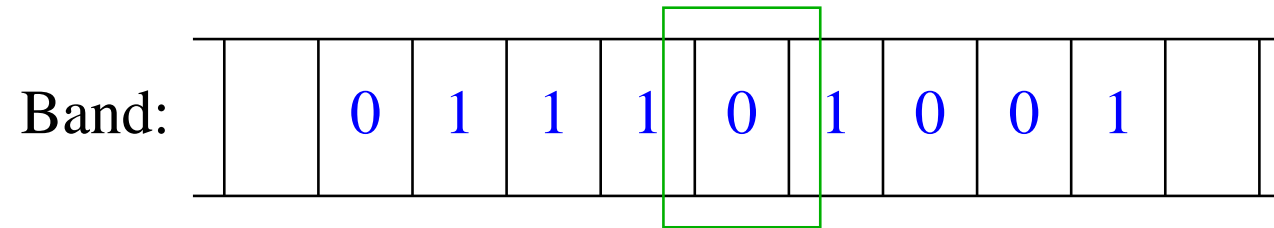


Alan Turing, 1912–1954

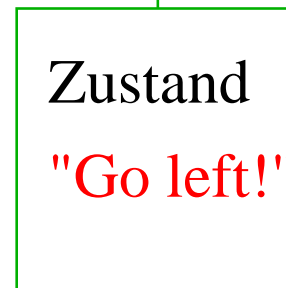
Daten der Turing-Maschine: Eine Folge von 0 und 1 und evt. weiterer Symbole wie z.B. " " (**Blank** – Leerzeichen) auf einem **Band** zusammen mit einer Position des "Schreib/Lese"-Kopfs auf dem Band;

Operationen: Überschreiben des aktuellen Zeichens und Verrücken des Kopfs um eine Position nach rechts oder links;

Kontrollstrukturen: Es gibt eine endliche Menge Q von **Zuständen**. In Abhängigkeit vom aktuellen Zustand und dem gelesenen Zeichen wird die Operation ausgewählt – und der Zustand geändert.

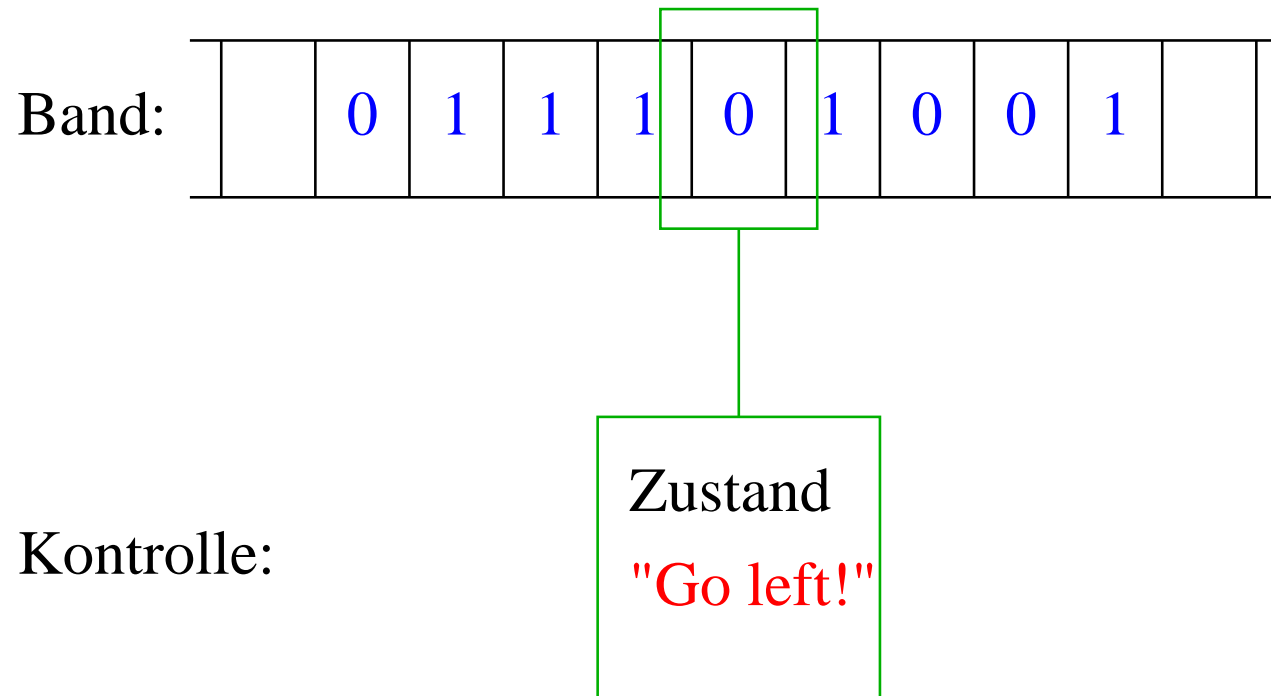


Kontrolle:



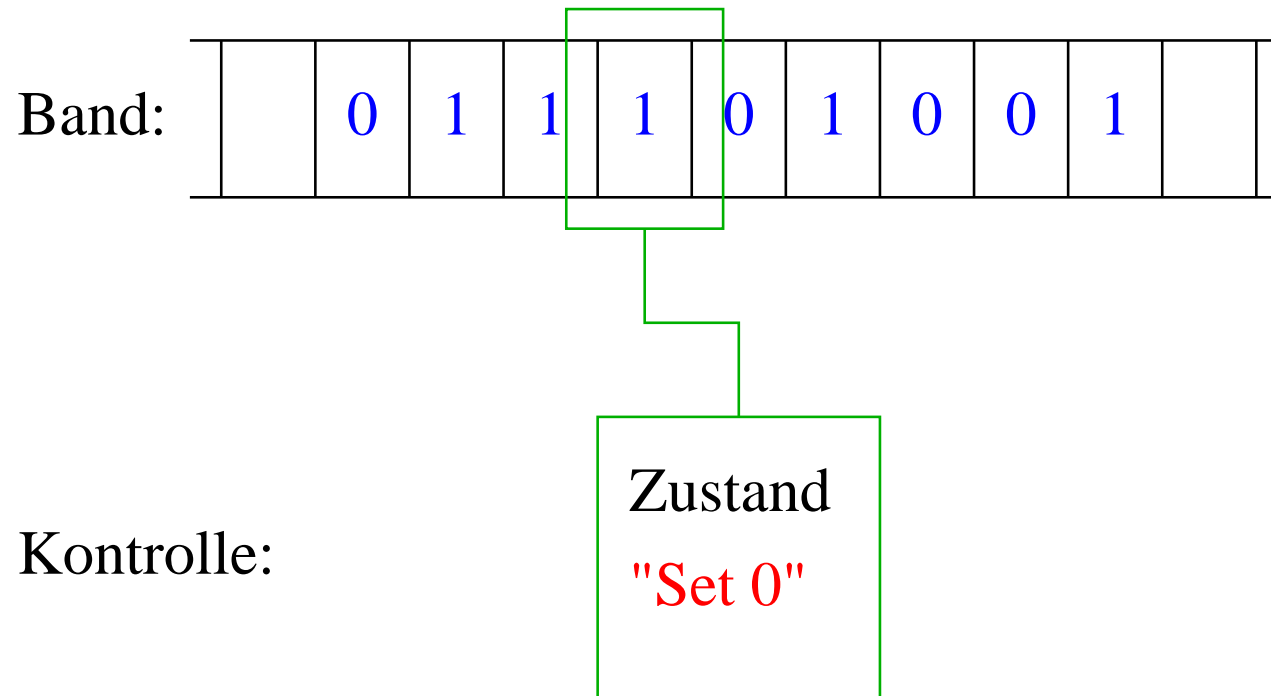
Programm:

Zustand	Input	Operation	neuer Zustand
"Go left!"	0	0 links	"Set 0"
"Go left!"	1	1 rechts	"Go left!"
"Set 0"	0	0 –	"Stop"
"Set 0"	1	0 links	"Set 0"



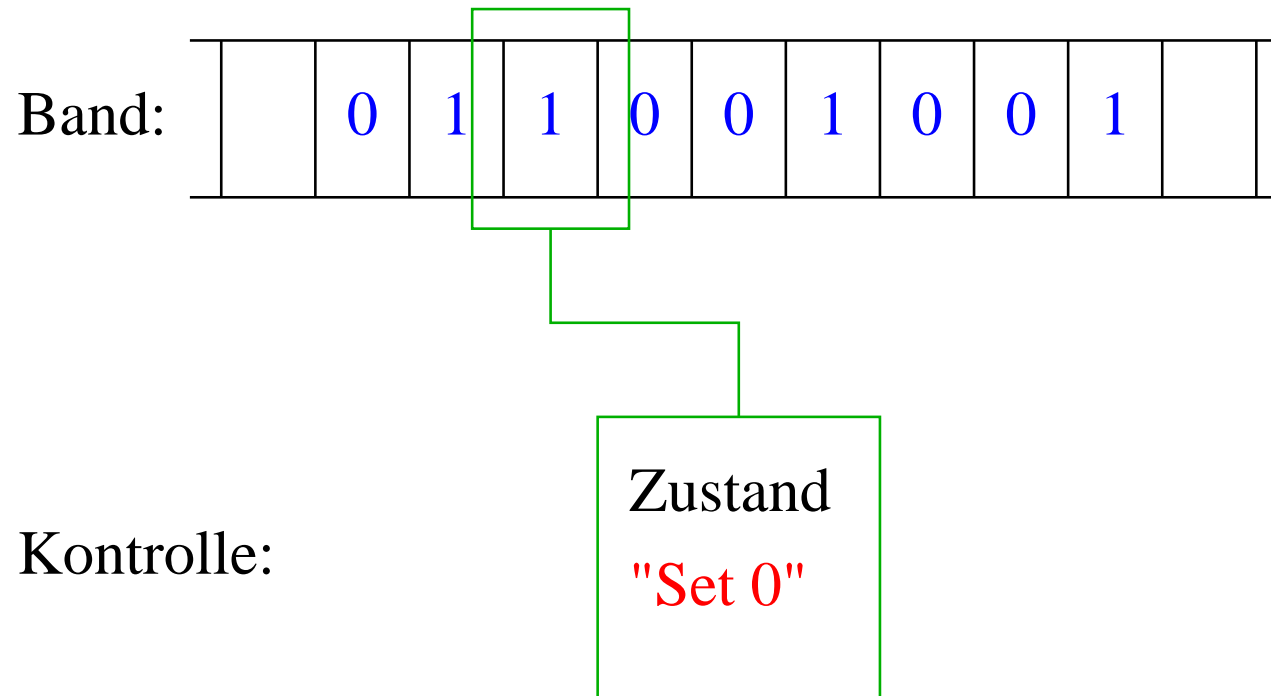
Operation = "Schreibe eine 0 und gehe nach links!"

neuer Zustand = "Set 0"

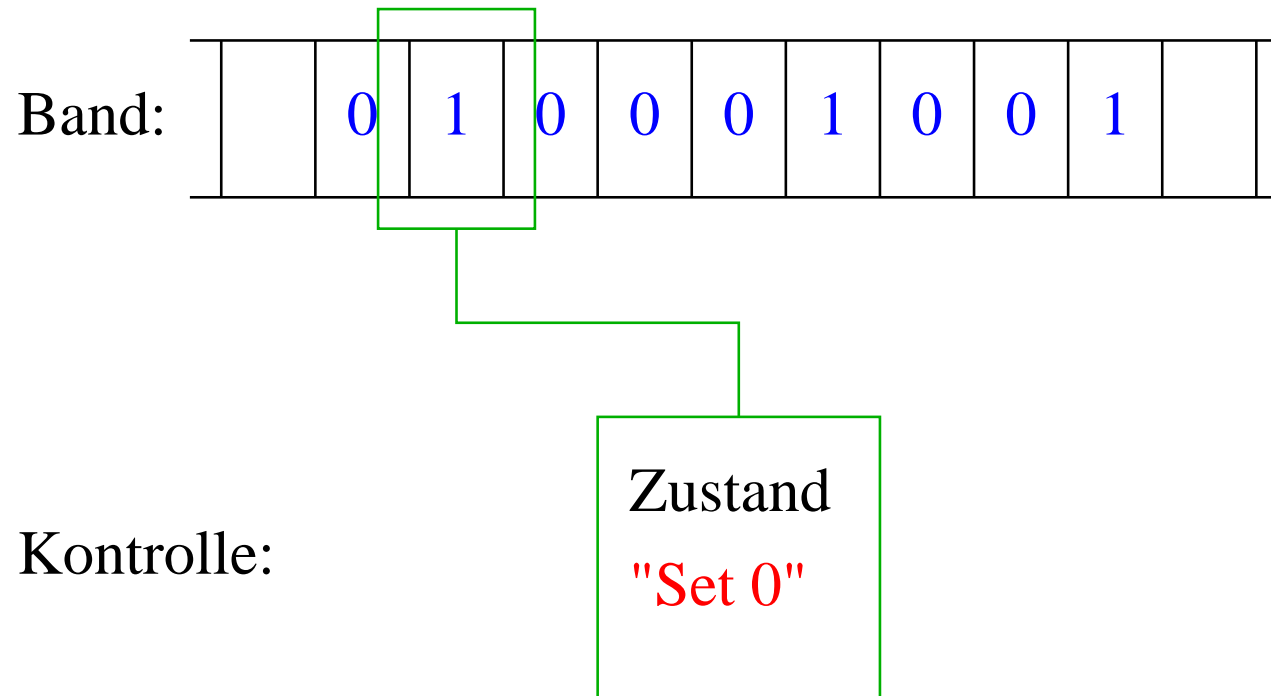


Operation = "Schreibe eine 0 und gehe nach links!"

neuer Zustand = unverändert

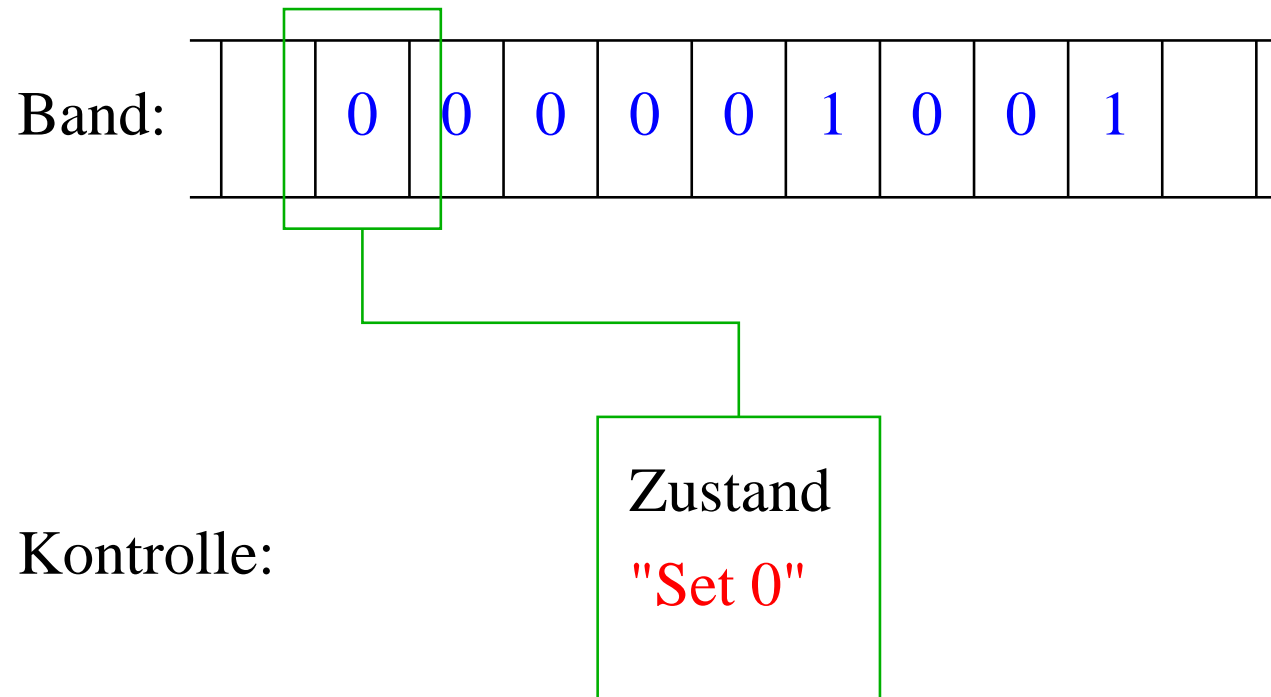


Operation = "Schreibe eine 0 und gehe nach links!"
neuer Zustand = unverändert

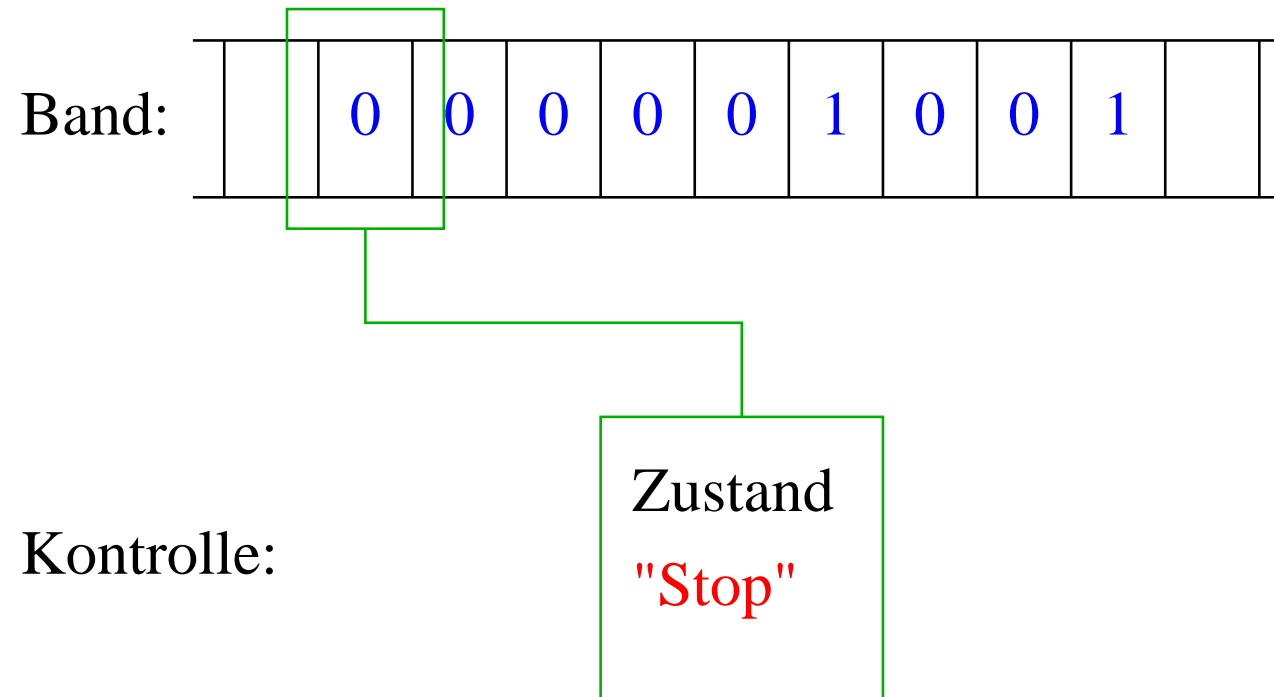


Operation = "Schreibe eine 0 und gehe nach links!"

neuer Zustand = unverändert



Operation = keine
neuer Zustand = "Stop"



Ende der Berechnung.

Fazit:

Die Turing-Maschine ist

- ... sehr einfach;
- ... sehr mühsam zu programmieren;
- ... aber nichtsdestoweniger **universell**, d.h. prinzipiell in der Lage **alles** zu berechnen, d.h. insbesondere alles, was ein Aldi-PC kann :-)

⇒ beliebtes Hilfsmittel in der ↑**Berechenbarkeitstheorie** und in der ↑**Komplexitätstheorie**.

Beispiel 2: JVM

- minimale Menge von Operationen, Kontroll- sowie Datenstrukturen, um **Java**-Programme auszuführen.
 - ⇒ Um **Java** auf einem Rechner XYZ auszuführen, benötigt man nur einen Simulator für die **JVM**, der auf XYZ läuft.
 - ⇒ **Portabilität!**

Ähnliche abstrakte Maschinen gibt es auch für viele andere Programmiersprachen, z.B. **Pascal, SmallTalk, Prolog, SML,...**

↑ **Compilerbau**

2 Eine einfache Programmiersprache

Eine Programmiersprache soll

- Datenstrukturen anbieten;
- Operationen auf Daten erlauben;
- **Kontrollstrukturen** zur Ablaufsteuerung bereit stellen.

Als Beispiel betrachten wir **MiniJava**.

2.1 Variablen

Um Daten zu speichern und auf gespeicherte Daten zugreifen zu können, stellt **MiniJava Variablen** zur Verfügung. Variablen müssen erst einmal eingeführt, d.h. **deklariert** werden.

Beispiel:

```
int x, result;
```

Diese Deklaration führt die beiden Variablen mit den **Namen** x und result ein.

Erklärung:

- Das Schlüsselwort `int` besagt, dass diese Variablen ganze Zahlen (“Integers”) speichern sollen.
`int` heißt auch **Typ** der Variablen `x` und `result`.
- Variablen können dann benutzt werden, um anzugeben, auf welche Daten Operationen angewendet werden sollen.
- Die Variablen in der Aufzählung sind durch Kommas “,” getrennt.
- Am Ende steht ein Semikolon “;”.

2.2 Operationen

Die Operationen sollen es gestatten, die Werte von Variablen zu modifizieren. Die wichtigste Operation ist die [Zuweisung](#).

Beispiele:

- `x = 7;`
Die Variable `x` erhält den Wert 7.
- `result = x;`
Der Wert der Variablen `x` wird ermittelt und der Variablen `result` zugewiesen.
- `result = x + 19;`
Der Wert der Variablen `x` wird ermittelt, 19 dazu gezählt und dann das Ergebnis der Variablen `result` zugewiesen.

- `result = x - 5;`

Der Wert der Variablen `x` wird ermittelt, 5 abgezogen und dann das Ergebnis der Variablen `result` zugewiesen.

Achtung:

- **Java** bezeichnet die Zuweisung mit "=" anstelle von " :=" (Erbschaft von **C** ... :-)
- Jede Zuweisung wird mit einem Semikolon ";" beendet.
- In der Zuweisung `x = x + 1;` greift das `x` auf der rechten Seite auf den Wert **vor** der Zuweisung zu.

Weiterhin benötigen wir Operationen, um Daten (Zahlen) einlesen bzw. ausgeben zu können.

- `x = read();`
Diese Operation liest eine Folge von Zeichen vom Terminal ein und interpretiert sie als eine ganze Zahl, deren Wert sie der Variablen `x` als Wert zu weist.
- `write(42);`
Diese Operation schreibt 42 auf die Ausgabe.
- `write(result);`
Diese Operation bestimmt den Wert der Variablen `result` und schreibt dann diesen auf die Ausgabe.
- `write(x-14);`
Diese Operation bestimmt den Wert der Variablen `x`, subtrahiert 14 und schreibt das Ergebnis auf die Ausgabe.

Achtung:

- Das Argument der `write`-Operation in den Beispielen ist ein `int`.
- Um es ausgeben zu können, muss es in eine **Folge von Zeichen** umgewandelt werden, d.h. einen `String`.

Damit wir auch freundliche Worte ausgeben können, gestatten wir auch **direkt** `Strings` als Argumente:

- `write("Hello World!");`
... schreibt `Hello World!` auf die Ausgabe.

2.3 Kontrollstrukturen

Sequenz:

```
int x, y, result;  
x = read();  
y = read();  
result = x + y;  
write(result);
```

- Zu jedem Zeitpunkt wird nur eine Operation ausgeführt.
- Jede Operation wird genau einmal ausgeführt. Keine wird wiederholt, keine ausgelassen.
- Die Reihenfolge, in der die Operationen ausgeführt werden, ist die gleiche, in der sie im Programm stehen (d.h. nacheinander).
- Mit Beendigung der letzten Operation endet die Programm-Ausführung.

⇒ Sequenz alleine erlaubt nur sehr einfache Programme.